1.

The aim of the project was to identity the Persons Of Interest(POIs) involved in the Enron scam of 2002.

With a highly public and populated dataset, it makes sense to identify POI using the features of the dataset. To that extent, machine learning can help us predict whether a person in the Enron scam was a POI or not. The advantage of machine learning is that we can use this mediocrely sized corpus to select some finely tuned discriminatory factors to give to the machine so that it can create a kind of 'persona' of POIs (their common factors) and identify them easily.

Some facts about the dataset:

# of people: 146

# of POI: 18

# of non-POI: 128

Features with NaN: almost all, with a lot in director\_fees

# of features: 18 (existing) + 2 (created)

Some features: financial - salary, bonus, long term incentives, deferred income; personal - email, poi, name

Outliers: The field 'TOTAL' is an outlier because it most probably arose from a spreasheet typo: 'TOTAL' is definitely not a person at all! The reason I found it was by visualizing data, and finding the maximum from the dataset and deleting from there.

The one other major outlier led to discover a powerful opponent of the scams, Vincent Kaminiski. More information in the comments.

Besides these, there were no other significant outliers.

2.

**INTRODUCTION**

Given the vast number of options, I decided to select a few for simplicity and convenience, besides saving runtime. Also, I was confident that some features have more importance than others; in effect, they 'drive' the analysis; I wanted such variables. I plotted the POI variable against most of the numerical variables to see any trends or interesting features. Further, I decided to combine 2 numerical features if they collectively gave me information on the POIs. Visualizations really helped me see the differences in value. In the end, I used the following 4 variables:

(salary, long\_term\_incentive, bonus/salary, high\_from\_to)

**INDEPTH REASONS FOR SELECTION OF MY FEATURES**

The features chosen were so because they exhibited the following characteristics:

1. There was a clear-cut positive/negative trend of the data wrt to this feature and POIs/non-POIs
2. [IMPORTANT] There were enough data points wrt to the feature values such that it would be easy to discriminate between POIs and non-POIs.

There were high direct correlations for each of ‘salary’ and ‘long\_term\_incentive’. This made sense given that the people involved in the Enron scam would have ensured that they got the most money for the buck and that their long-term future was well-planned too.

While plotting my data my data, it seemed like plotting bonus vs salary didn't show any trends; furthermore, plotting plotted salary vs bonus as a %age of the salary showed much more! This is because it was easier to see employees whose bonus exceeded their salary by more than 10 times!  
There was definitely something fishy with the fact that a pretty large percentage of these people were POIs. It made sense, thus, to create a feature called ‘bonus\_salary’ that measured persons with high salary and enormous bonuses!

Similarly, I found senders/recipients of high 'from' and 'to' emails were often POI. Thus, 'high\_from\_to' was another feature I created as quite a lot of POIs were involved in sending a lot of emails from and to each other.

**WHY WERE OTHER FACTORS NOT TAKEN INTO ACCOUNT**  
Other features such as ‘other’ and ‘total\_stock\_value’ did not seem to have a high correlation with the POI – all such observations have been made using the graphs. Through the visualization of the graphs, I could see whether there was a clear trend in the data; this is often much easier than using other analytic measures.

It makes sense that factors such as ‘director’s fees’ and ‘restricted\_stock\_deferred’ were not very helpful in the sense that most of these values were NaN.

Thus, it was through various graphs, manual selection and good intuition and testing that I finally arrived at my features.

**FEATURE SCALING**

There was no need to scale features in the categorical variables and the numerical variables were within range of one another to prevent any scaling here as well.

**TABLE OF METRICS REGARDING DIFFERENT FEATURES**

Here are the details in tabular format:

|  |  |  |  |
| --- | --- | --- | --- |
| Feature list | Addition of feature | New precision | New recall |
| - | - | 0 | 0 |
| Salary | salary | 0.26 | 0.32 |
| Salary, long term incentive | Long term incentive | 0.30 | 0.38 |
| Salary, long term incentive, high\_from\_to | High\_from\_to | 0.32 | 0.43 |
| Salary, long term incentive, high\_from\_to, salary bonus | **Salary\_bonus** | **0.33** | **0.45** |
|  |  |  |  |
| Salary, long term incentive, high\_from\_to, salary bonus, deferred income | Deferred income | 0.30 | 0.34 |
| Salary, long term incentive, high\_from\_to, salary bonus, total payments | Total payments | 0.31 | 0.35 |
| Salary, long term incentive, high\_from\_to, salary bonus, total\_stock\_value | Total stock value | 0.30 | 0.36 |
| - | - | - | ­- |
| Salary | Salary | 0.26 | 0.32 |
| Salary, total stock value | Total stock value | 0.30 | 0.38 |
| Salary, total stock value, from\_messages | From\_messages | 0.28 | 0.36 |
| …. | …. | …. | … |
| Deferred\_income, other, from\_this\_person\_to\_poi | ---- | 0.30 | 0.40 |
| Salary, other, long\_term\_incentive | ---- | 0.26 | 0.39 |
| Bonus, other, salary\_bonus | ---- | 0.16 | 0.27 |
| long\_term\_incentive, salary\_bonus, from\_messages | ---- | 0.20 | 0.30 |

**RESULT OF ADDING SELF-CREATED VARIABLES**

From the details above, you can see it is that these 2 new features in fact added to the precision and recall (refer to first 5 rows and bolded row above).

**RESULT OF USING OTHER SETS OF VARIABLES**

Refer to rows below the first 5 rows.

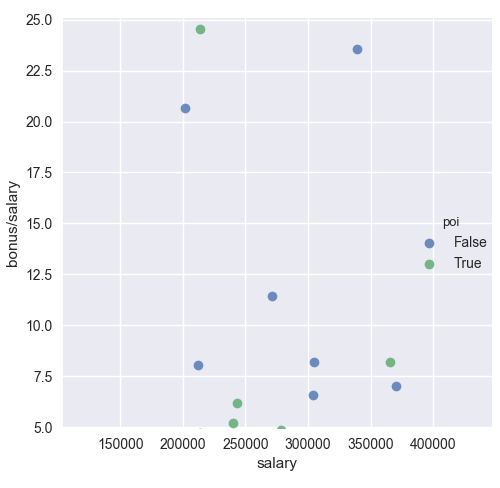
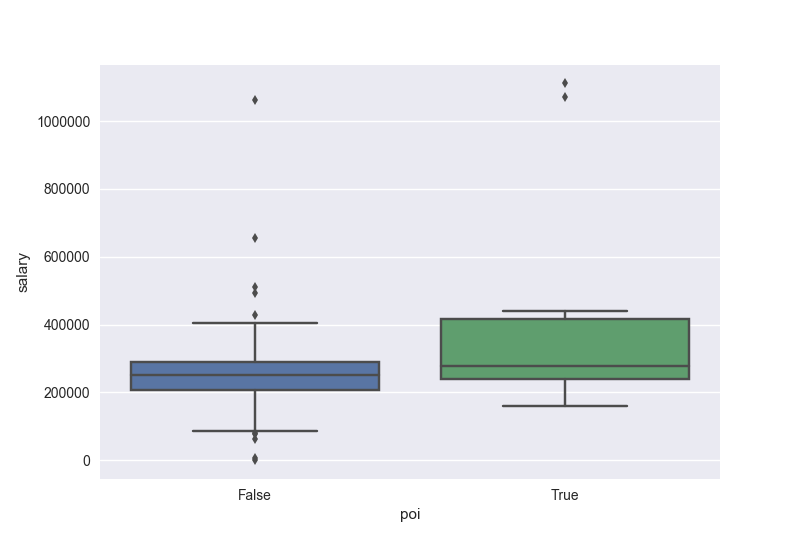
Also from above, you can see that adding new feature only decreases the overall precision and recall.

Also note how different feature sets only result in poor precision and/or recall or both!

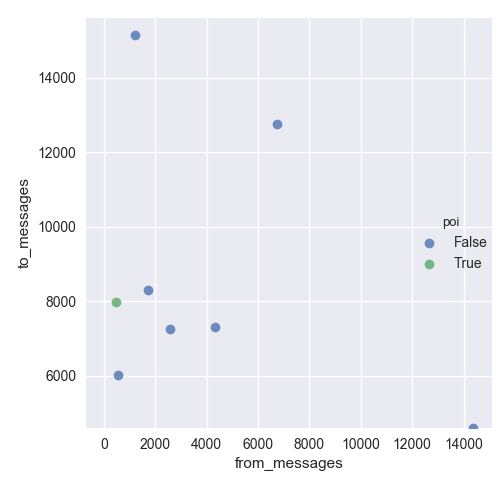
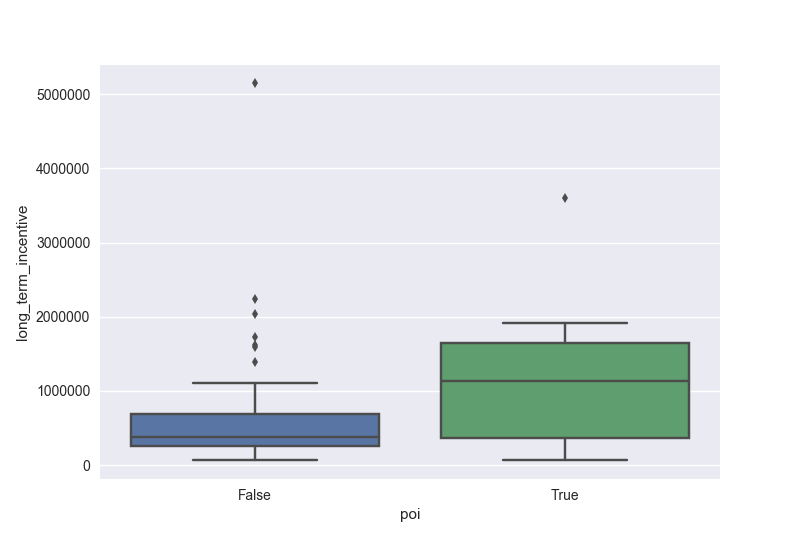
Plots/Graphs on next page!

**PLOTS/GRAPHS**

Bonus/Salary Salary



Long term incentive High from/to messages



In the case of the boxplots, it is apparent that the plot for POIs in both cases is much higher, and thus it makes sense to use these.

In the case of the scatterplots, there are enough points, and a majority the same type(POI/non-POI).

**FEATURE IMPORTANCES**

Here are the feature importances:

[ 0.04122 0.03254 0.82105 0.1089]

It seemed, even in the variables I chose, some were more powerful than others.

3.

I tested my data on all 4 algorithms - Naive Bayes, SVM, Decision Trees and Random Forests. Of these, SVMs didn't seem to work and Decision Trees worked best, even though I know the perfect accuracy of 1.0 when testing on the training data was an overfit. Overall, when there was a train-test split (cross validated, K fold = 1000), even though the decision tree's accuracy was 4 and 2 percent lower than the 1st and 2nd algorithms, it's precision and recall were so MUCH better, leading to a higher F1 score.

The final values for all algorithms were:

-> GaussianNB(priors=None)

Accuracy: 0.84167 Precision: 0.32227 Recall: 0.17000 F1: 0.22259 F2: 0.18774

Total predictions: 15000 True positives: 340 False positives: 715 False negatives: 1660 True negatives: 12285

-> DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

presort=False, random\_state=None, splitter='best')

Accuracy: 0.80627 Precision: 0.33197 Recall: 0.44750 F1: 0.38118 F2: 0.41838

Total predictions: 15000 True positives: 895 False positives: 1801 False negatives: 1105 True negatives: 11199

-> RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=10, n\_jobs=1, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False)

Accuracy: 0.82167 Precision: 0.32980 Recall: 0.32700 F1: 0.32840 F2: 0.32756

Total predictions: 15000 True positives: 654 False positives: 1329 False negatives: 1346 True negatives: 11671

As can be seen, the Decision Tree's other metrics MORE than make up for the lack of accuracy!

4.

Changing the parameters of an algorithm can make a lot of difference to the alogrithm's performance. Each situation has its own set of parameters that suit it best. This is called tuning the parameters of the algorithm. Not tuning the algorithm can result in a poorer performance than can be achieved. For example:

To tune the Decision Tree, I looked at SKLearn's documentation of Decision Trees and combined it with the knowledge I gained in the class to tune the following parameters to the following values:

{'max\_features': None, 'min\_impurity\_split': 1e-07, 'criterion': 'gini'}

I used GridSearchCV to choose from the best set of values for these parameters:

parameters = {'criterion':('gini', 'entropy'), 'max\_features': [None, 1, 2, 3], 'min\_impurity\_split': [1e-7, 1e-3, 1e1]}

Here are the results

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

presort=False, random\_state=None, splitter='best')

Accuracy: 0.80560 Precision: 0.33137 Recall: 0.45000 F1: 0.38168 F2: 0.41993

Total predictions: 15000 True positives: 900 False positives: 1816 False negatives: 1100 True negatives: 11184

As you can see, it did increase metrics by a little bit.

5.

**VALIDATION: DEFINITION & PURPOSE**

In machine learning, validation is referred to as the process where a trained model is evaluated with a testing data set. The testing data set is a separate portion of the same data set from which the training set is derived. The main purpose of using the testing data set is to test the generalization ability of a trained model.

**A CLASSIC MISTAKE**

A classic mistake you commit (which my Decision Tree did) by using the same data set to validate your algorithm is to over-fit your data (high bias) - this happens because your algorithm already knew about the data you tested it with, so it WILL high you a high score on accuracy (in my case - I got a perfect accuracy of 1.0, something that is nearly impossible!).

**tester.py**

To verify my classifier, a tester.py script has been provided that takes a dataset, classifier and a list of features and outputs metrics such as the parameters of the classifier and several metrics including accuracy, precision, recall, F1 scores and number of true positives, false negatives, etc. My script imports the Python file containing this function.

To validate your analysis, you can split your data into test and train (cross validation), or even better, use an approach known as stratified (stratification is the arrangement or classification of something into different groups) shuffle**:** a way to create copies of the data with different training and testing data each time for a more generalized metric of validation. A fold is basically an iteration of the data; it is done multiple times to provide a more generalized and accurate answer. Here 1000 iterations are performed.

The Python model, StratifiedShuffleSplit used in the tester script is a merge of StratifiedKFold and ShuffleSplit, which returns stratified randomized folds. **The folds are made by preserving the percentage of samples for each class.** This is the main idea so that each subset has each an appropriate number of training points. **All of this is employed by my script when it calls the testing function in ‘tester.py’**

6.

Two metrics used were precision and recall.

The precision was 0.33 and the recall was 0.45.

This means, on average, that my algorithm identified 45% of all possible POI, and that 33% of the time my algorithm correctly identified a POI when it was a POI.